# Compositional Verification of Parameterised Timed Systems

Lacramioara Astefanoaiei, Souha Ben Rayana, Saddek Bensalem, Marius Bozga, Jacques Combaz

# Compositional Verification of Parameterised Timed Systems

L. Aştefănoaei, S. Ben Rayana, S. Bensalem, M. Bozga, J. Combaz

Univ. Grenoble Alpes, VERIMAG, F-38000 Grenoble, France
CNRS, VERIMAG, F-38000 Grenoble, France **

**Abstract.** In this paper we address the problem of *uniform* verification of parameterised *timed* systems (PTS): "*does a given safety state property hold for a system containing n identical timed components regardless of the value of n?*". Our approach is compositional and consequently it suits quite well such systems in that it presents the advantage of reusing existing local characterisations at the global level of system characterisation. Additionally, we show how a direct consequence of the modelling choices adopted in our framework leads to an elegant application of the presented method to topologies such as stars and rings.

## 1 Introduction

Swarm robots, satellite systems, multithreaded programs, ad-hoc networks, device drivers, all these applications have in common a structural characteristic: they rely on multiple copies of the same program interacting between each other, that is, they constitute systems *parameterised* by some components which are being replicated. Though the individual "replicas" may not involve a too complicated code in itself, the systems containing them are quite complex. The inherent complexity has several sources: it may come from that the systems are considerably large, as it is the case for swarm robots, or from that, effectively, their size cannot be a priori known, as it is the case for satellites. Yet another delicate matter is that these systems are highly dynamic and adaptable as their topology may change depending on initial goals, component failures, etc. All in all, the verification of such parameterised systems reveals real challenges.

There is an extensive amount of work on the verification of *untimed* parameterised systems. Some [22, 17, 32, 31, 14, 26] focus on particular classes for which the problem of uniform verification is decidable. Among these classes, we name *well-structured transition systems* [20, 1, 21] for which decidability follows from the existence of a so-called *well-quasi ordering* between states. Two examples that fit this class are Petri nets, and lossy channel systems. Most notably, the work in [32] shows that, for *bounded-data parameterised systems* and for a restricted fragment of properties there is always a small number $n$ (the so-called

** Work partially supported by the European Projects 257414 ASCENS, STREP 318772 D-MILS, French BGLE Manycorelabs, and Artemis AIPP Arrowhead.

*cutoff*, later leading to *small model theorems*) such that if one can show correctness for the systems with less than $n$ replicas then the system itself is correct. Others focus on *incomplete* but general methods: semi-automatic approaches based upon explicit induction [18] and upon network invariants [34, 4, 29, 30] or automatic ones based upon abstraction [11, 12], upon regular model-checking (for a survey [6]), or upon symmetry reduction [19].

In this paper we address the problem of uniform verification of parameterised *timed* systems (PTS). The existing approaches are less numerous than for untimed systems. The work in [5, 3] concentrates on decidability: the authors show the decidability of the reachability of PTSs where processes are timed automata with either only one clock or otherwise time is discrete. These results have been later generalised to timed ad hoc networks in [2] where it is shown that even for processes as timed automata with one clock, for *star topologies* where the diameter between nodes is of length 5, the reachability problem is undecidable. It is also the case for processes with 2 clocks and *clique topologies*. Decidability holds for special topologies as stars with diameter 3 and cliques of arbitrary order for processes as timed automata with 1 clock. For discrete time, reachability is decidable for any number of clocks and topologies as graphs with bounded paths. As a side remark, all the positive results above rely on the technique of well-quasi-orderings mentioned above. The approaches in [27, 16, 15] are closer in spirit to ours. The work in [16, 15] shows how reachability of parameterised systems where processes are timed automata can be encoded as a formula in a decidable fragment of the theory of arrays [23]. The work in [27] concentrates upon parameterised *rectangular* hybrid automata nets. They show *a small model theorem* for such systems. The proof typically follows the lines from the one showing the existence of cutoffs in [32].

Our approach borrows from [27] and builds upon the methodology described in [10]. There, a compositional method is introduced for the verification of fixed size timed systems. It did so by locally computing invariants for each component and for their interactions and checking with an SMT solver, Z3[1], if their conjunction implies the validity of a given safety property. The method being compositional suits quite well parameterised systems in that it presents the advantage of reusing existing local characterisations at the global level of system characterisation. Applying the method in the context of PTSs boils down to giving an effective method of checking the validity of quantified formulae. This is not obvious because, for instance, Z3 fails while trying to instantiate it to disprove it. At a first thought, one could apply some tactics which make extensive use of transitivity and practically reduce the formula to a tautology. However, to do this, one would need to transform the initial formula into disjunctive normal form, which is costly. At a second thought, following the reasoning from [15], we could show that the formula we feed to Z3 fits well in the theory of arrays. However, a simpler and more inspired solution is to make use of the small model theorem from [27]. The advantage of combining such result with the compositional method from [10] is twofold. On the one hand it can be the

[1] rise4fun.com/Z3

case that the system without replicas is big enough to make the construction of the product infeasible. On the other hand, a direct consequence of the modelling choices adopted in our framework leads to an elegant application of the presented method to parameterised timed systems where interactions are given by various types of topologies which extend the standard binary synchronous communication from [27]. With respect to the work in [15], our formulae are quite small while there the resulting formulae have a number of quantifiers proportional to the length of the fixpoint computation of the reachability set.

*Organisation of the paper.* Section 2 recalls the needed existing results. Section 3 introduces the semantics of PTSs while Section 4 shows how to effectively verify PTSs compositionally. Section 5 describes two applications on classical examples and Section 6 concludes.

## 2 Preliminaries

Following [10], our method builds upon the verification rule (VR) from [13]. Assume that a system consists of $n$ components $B^i$ interacting by means of an interaction set $\gamma$, and that the property that the system should satisfy is $\Psi$. If components $B^i$ and interactions $\gamma$ can be locally characterised by means of invariants (here denoted $CI(B^i)$, resp. $II(\gamma)$), and if $\Psi$ can be proved to be a logical consequence of the conjunction of the local invariants, then $\Psi$ is a global invariant. In the rule (VR) depicted in Figure 1 the symbol " $\vdash$ " is used to underline that the logical implication can be effectively proved (for instance with an SMT solver) and the notation "$B \models \Box\, \Psi$" is to be read as "$\Psi$ holds in every reachable state of $B$".

$$\frac{\vdash \bigwedge_i CI(B^i) \wedge II(\gamma) \rightarrow \Psi}{\|_\gamma B^i \models \Box\, \Psi} \text{ (VR)}$$

**Fig. 1.** Compositional Verification

The method in [10] extends, in a modular manner, the above rule with the purpose of applying it to the verification of timed systems. The framework in this paper is that of *parameterised* timed systems. We show how compositional verification along the lines of the methodology of [10] works for parameterised timed systems. Before, we recall the standard concepts we make use of.

**Timed automata.** We use timed automata (TA) to represent the behaviour of components. Timed automata have control locations and transitions between these locations. Transitions may have timing constraints, which are defined on clocks. Clocks can be reset and/or tested along with transition execution. Formally, a timed automaton is a tuple $(L, \Sigma, T, X, \mathsf{tpc}, s_0)$ where $L$ is a finite set of control locations, $\Sigma$ a finite set of actions, $X$ is a finite set of clocks, $T \subseteq L \times (\Sigma \times \mathcal{C} \times 2^X) \times L$ is finite set of transitions labelled with actions, guards, and a subset of clocks to be reset, and $\mathsf{tpc} : L \rightarrow \mathcal{C}$ assigns a time progress condition to each location. $\mathcal{C}$ is the set of clock constraints and $s_0 \in L \times \mathcal{C}$ provides the initial configuration. Clock constraints are conjunctions of (in)equalities of the form $x \# ct$ or $x - y \# ct$ with $x, y \in \mathcal{X}$, $\# \in \{<, \leq, =, \geq, >\}$ and $ct \in \mathbb{Z}$. Time progress conditions are restricted to conjunctions of constraints as $x \leq ct$.

A timed automaton is a syntactic structure whose semantics is based on continuous and synchronous time progress. A state of a timed automaton is given by a control location paired with real-valued assignments of the clocks. From a given state, a timed automaton can let time progress when permitted by the time progress condition of the corresponding location, or can execute a (discrete) transition if its guard evaluates to true. The effect of time progress of $\delta \in \mathbb{R}^+$ units of time is to increase synchronously all clocks by $\delta$. Executions of transitions are instantaneous: they keep values of clocks unchanged except the ones that are reset (i.e., assigned 0). Because of their continuous semantics, most timed automata have infinite state spaces. However, they admit finite symbolic representations of their state spaces as the so-called *zone* graphs [8, 7, 25, 35].

**T-assertions.** We use T-assertions to express local and system properties. This choice is motivated by the fact that, following a result from [27], the validity of T-assertions is decidable. T-assertions are a particular[2] case of LH-assertions. The signature of T-assertions consists of the constants 1 and $n$ of type $\mathbb{N}$, and of a finite number of variables: (a) *index variables*: $i_1, \ldots, i_a \in \mathbb{N}$; (b) *discrete variables*: $l_1, \ldots, l_b \in L$; (c) *real variables*: $x_1, \ldots, x_c \in \mathbb{R}$; (d) *discrete array variables*: $\bar{l}_1, \ldots, \bar{l}_d : [n] \to L$; (e) *real array variables*: $\bar{x}_1, \ldots, \bar{x}_e : [n] \to \mathbb{R}^+$ where by $[n]$ we denote the set $\{1, \ldots, n\}$. Terms are given by the BNF grammar:

$$\texttt{ITerm} ::= 1 \mid n \mid i_j \qquad \texttt{DTerm} ::= L_j \mid l_k \mid \bar{l}_j[\texttt{ITerm}] \qquad \texttt{RTerm} ::= x_j \mid \bar{x}_k[\texttt{ITerm}]$$

and the formulae are structurally defined as:

$$\texttt{Atom} ::= \texttt{ITerm} < \texttt{ITerm} \mid \texttt{DTerm} = L_k \mid a \cdot \texttt{RTerm} + b \cdot \texttt{RTerm} + c < 0$$
$$\texttt{Formula} ::= \texttt{Atom} \mid \neg\texttt{Formula} \mid \texttt{Formula} \wedge \texttt{Formula}$$

with $a, b, c \in \mathbb{R}$. T-assertions are of the form $\forall i_1, \ldots, i_k \in [n]\ \exists j_1, \ldots, j_m \in [n].F$ where $F$ is of type `Formula`. We note that equality and non strict comparisons between indices and real variables can be expressed by means of $\wedge, \neg, <$. For example, $i = j$ is written as $\neg(i < j) \wedge \neg(j < i)$. It is also the case that addition with constants can be expressed by means of extra quantifiers. For example, $j = i + 1$ is written as $i < j \wedge \forall k.i < k \to j \leq k$. This construction generalises to $j = i + o$ for $o$ an integer constant. To restrict indices within bounds, we make the convention that addition is understood modulo $n$. For succinctness, in the rest of the paper we adopt the notation $\overline{x}[i + o]$ to stand for $\exists j.j = i + o \wedge \overline{x}[j]$.

*Example 1.* The following T-assertions express safety properties:

1. $\forall i \neq j.\neg(\bar{l}[i] = C \wedge \bar{l}[j] = C)$ expresses mutual exclusion for $C$ denoting that a process is in the critical location;
2. $\forall i, j.(\bar{l}[i] = \bar{l}[j] \to\mid \bar{x}[i] - \bar{x}[j] \mid< 6)$ expresses a "maximum delay" between the timings of any two processes which are in the same location.

[2] To be specific, by "particular" we mean that we do not need the so called "index-valued array variables" which in [27] model pointer variables.

As in [27], the semantics of a T-assertion $\Phi$ is given by $n$-models, denoted as $M(n, \Phi)$, which interpret the index, the discrete and resp. the real variables in $\Phi$ as taking values in $[n]$, $L$, and resp. $\mathbb{R}^+$.

*Example 2.* A 2-model for the mutual exclusion in Example 1 is $\bar{l} = [C, I]$ where say $I$ denotes idle locations. A 4-model for the maximum delay property is $\bar{l} = [L_1, L_2, L_1, L_2], \bar{x} = [10, 8, 6, 3]$.

T-assertions have a *small model theorem*. This is a key fact that can be exploited for automatic verification in general.

**Proposition 1 (Simplified from [27])** *Let $\Phi$ be a T-assertion given in the form $\forall i_1, \ldots, i_k \in [n]\ \exists j_1, \ldots, j_m \in [n].\phi$ where $\phi$ is a quantifier-free formula involving the index variables $i_1, \ldots, i_k$, $j_1, \ldots, j_m$ and array variables. We have that $\Phi$ is valid iff, for all $n \leq k + 2$, $\Phi$ is satisfied by all $n$-models.*

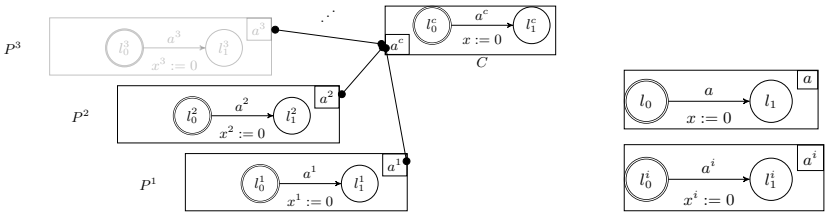
Next, we introduce our formalisation of PTSs and show how we can take advantage of the small model result to compositionally verify PTSs.

## 3 PTSs and their Semantics

In our framework, PTSs are understood as consisting of possibly (but not also necessarily) a *fixed* number of components and an *arbitrary* number $n$ of isomorphic processes $P^i$ all given as TAs and interacting by means of an interaction set $\gamma$. In what follows, we adopt the notations $C$ for the non parameterised part of a PTS, $C \|^n_\gamma P^i$ for a PTS itself. For ease of reference, we use $\Sigma_C$, $\Sigma$, $\Sigma^i$ to denote the actions of $C$, of a generic process $P$ and of a process $i$, $P^i$. $\Sigma^i$ is obtained from $\Sigma$ by attaching $i$ to each action in $\Sigma$. An example of a PTS is depicted in Figure 2a. $C$ interacts with some processes $P^i$ by synchronising actions $a$ and $a^i$ while resetting clocks $x^c$ and $x^i$. As Figure 2b illustrates, components $P^i$ are obtained from the same generic timed automaton $P$ consisting of two control locations $l_0$ and $l_1$ and one transition[3] from $l_0$ to $l_1$ labelled by action $a$ and resetting clock $x$. The construction of $P^i$ from a generic $P$ is straightforward: each location $l$, clock $x$, and action $a$ are mapped into $l^i$, $x^i$, $a^i$ respectively.

Components interact by means of strong synchronisation between their actions. The synchronisations are specified in the so called *interactions* as sets of actions. An interaction can involve at most one action of each component. For the ease of reference, the whole set of interactions is denoted by $\gamma$. In a *parameterised setting*, we define $\gamma$ as a set of *interaction patterns* instead. An interaction pattern $\alpha$ is a tuple $\big(a^c, (a_1, o_1), \ldots, (a_m, o_m)\big) \in \Sigma^c \times (\Sigma \times \mathbb{N})^m$ such that $0 = o_1 < o_2 < \cdots < o_m$. An interaction pattern describes at an abstract level a family of interactions between $C$ and $m$ processes[4]: $\big(a^c, (a_1, o_1), \ldots, (a_m, o_m)\big)$ generates $n$ interactions $\alpha^i = \big(a^c, a_1^{i+o_1}, \ldots, a_m^{i+o_m}\big) \in \Sigma^c \times \Sigma^{i+o_1} \times \cdots \times \Sigma^{i+o_m}$

[3] Non displayed guards and time progress conditions of locations are by default *true*.
[4] The case of PTSs without $C$ is similar and we illustrate it only by means of examples.




(a) $C\|^n_\gamma P^i$ (b) $P$ and $P^i$

**Fig. 2.** An Example of a PTS and the construction of $P^i$

where all sums are understood modulo $n$. We use $gen(\alpha)$ to denote the interactions generated by $\alpha$, that is, $\cup_{i\in[n]}\alpha^i$. By abuse of notation, we refer to $\gamma$ as either the set of interaction patterns or as $\cup_{\alpha\in\gamma} gen(\alpha)$, the set of all interactions generated by the patterns. The distinction should be clear from the context.

*Example 3.* The family of interactions $\{(a^c, a^1), (a^c, a^2), \ldots, (a^c, a^n)\}$ in Figure 2a is given by the interaction pattern $\alpha = \big(a^c, (a, 0)\big)$.

For $C$ as $(L^c, \Sigma^c, T^c, \mathcal{X}^c, \mathsf{tpc}^c, s_0^c)$, and $P^i$ as $(L^i, \Sigma^i, T^i, \mathcal{X}^i, \mathsf{tpc}^i, s_0^i)$, the semantics of $C\|^n_\gamma P^i$ is given by that of the timed automaton $(L, \gamma, T_\gamma, \mathcal{X}, \mathsf{tpc}, \mathbf{s_0})$ where $L = L^c \times_i L^i$, $\mathcal{X} = \mathcal{X}^c \cup_i \mathcal{X}^i$ and:

- for $s_0^c = (l^c, C_0)$, $s_0^i = (l_0^i, C_0^i)$, $\mathbf{s_0}$ is a pair of a global location $\mathbf{l_0} = (l_0^c, l_0^1, \ldots, l_0^n)$ and the initial clock constraints given by $C_0 \wedge_i C_0^i$,
- $\mathsf{tpc}\big((l^c, l^1, \ldots, l^n)\big) = \mathsf{tpc}(l^c) \wedge_i \mathsf{tpc}(l^i)$,
- for any $\alpha = \big(a^c, (a_1, o_1), \ldots, (a_m, o_m)\big)$ with $\alpha^i = \big(a^c, a_1^{i+o_1}, \ldots, a_m^{i+o_m}\big)$ and $\mathcal{O} = \{i + o_1, \ldots, i + o_m\}$, we have that:
  - if $l^c \xrightarrow{a^c, g^c, r^c} l'^c \in T^c$ and $l^i \xrightarrow{a^i, g^i, r^i} l'^i \in T^i$ for any $i \in \mathcal{O}$, $a^i \in \Sigma^i \cap \alpha^i$
  - then $(l^c, l^1, \ldots, l^n) \xrightarrow{\alpha^i, g, r} (l'^c, l'^1, \ldots, l'^n) \in T_\gamma$ with $l'^j = l^j$ for any $j \in [n] \setminus \mathcal{O}$ and $g = g^c \wedge_{i\in\mathcal{O}} g^i$, respectively $r = r^c \cup_{i\in\mathcal{O}} r^i$.

Interaction patterns have a considerable expressiveness power to the extent that they can encode regular topologies. Usually topologies are given by a graph where the vertices represent the indices of the processes and the edges give the communication between processes [2]. In our framework, the communication is given/induced by the set of interactions. There is a close correspondence between topologies and sets of interactions. This comes from the observation that topologies represented as graphs have a straightforward encoding as interaction sets. As an illustration, we consider the classical topology of a *ring*[5]. Given $n$ nodes, a ring topology naturally links a send from node $i$ to a receive at node $i + 1$, that is, it is generated by the pattern $((s, 0), (r, 1))$ where $s$, $r$ stand for "send", "receive". A graphical interpretation is given in Figure 3b.

[5] Rings are typically for binary communication, however broadcasts can be just as well encoded by means of interaction patterns.

*Example 4.* The interaction set in Figure 2a is generated by one pattern, namely $(a_c, (a, 0))$. The corresponding topology it describes is that of a *star*, another classical topology. The corresponding graphical depiction is in Figure 3a.

(a) Star. $\gamma = \{(a^c, (a, 0))\}$ (b) Ring. $\gamma = \{((s, 0), (r, 1))\}$

**Fig. 3.** Topologies & Interaction Sets

*Remark 1.* Thanks to the definition of $\gamma$ as a set, thus implicitly nondeterministic, with our method we cover any topology which may be enforced or hard-wired in the system at a later moment of time, or stage of design. To take a concrete example, the interaction set for the star topology does not oblige all components to participate, so any star "subset" (corresponding for instance to the situation when some components are turned off) is considered. This has the implication that, with respect to deadlock freedom, if our method yields true, the system is safe, then this is the case irrespectively of how many components are interacting.

## 4 Compositional Verification of PTSs

To compositionally verify PTSs, our method consists of automatically generating invariants characterising components, interactions and inter-component timings. These invariants are assembled in the (VR) rule recalled in the introduction. To apply the small model result from Proposition 1, the provided invariants need to be T-assertions. Next, we take them one by one and show how they can be effectively computed and shaped into the form of $T$-assertions.

### 4.1 Component Invariants

Component invariants characterise the reachable states of components when considered alone. Such invariants can be computed from the zones of the corresponding timed automata [8, 7, 25, 35]. More precisely, given that the set of the reachable symbolic states $(l_j, \zeta_j)$ of an arbitrary process $P$ is finite, its invariant is defined by the disjunction $\vee_j(l_j \wedge \zeta_j)$, where by abuse of notation $l_j$ is used to denote the predicate that holds whenever $P$ is at location $l_j$.

We recall that, for a process $P^i$, we identify locations and clocks as $l^i$, $x^i$, for locations $l$ and clocks $x$ in the generic process $P$. To fit the formulae characterising the reachable set of states of $P^i$ in the class of T-assertions, we indiscriminately view $l^i$ as the $i$th element in the array $\bar{l}$ and similarly, $x^i$ is equally viewed as the $i$th element in the array $\bar{x}$, that is, semantically we make no difference between $l^i$ and $\bar{l}[i]$, respectively $x^i$ and $\overline{x}[i]$.

*Example 5.* As an illustration, the component invariants for $C$, $P$ and $P^i$ (where $C$, $P^i$, $P$ are the ones depicted in Figures 2a,2b) are as follows:

$$CI(C) = (l_0^c \wedge x^c \geq 0) \vee (l_1^c \wedge x^c \geq 0) \quad (1)$$

$$CI(P) = (l_0 \wedge x \geq 0) \quad \vee \quad (l_1 \wedge x \geq 0)$$

$$CI(P^i) = (\overline{l_0}[i] \wedge \overline{x}[i] \geq 0) \vee (\overline{l_1}[i] \wedge \overline{x}[i] \geq 0) \quad (2)$$

We use $CI$ to denote the conjunction of $CI(C)$ and of all $CI(P^i)$. Extending the argument that the conjunction of invariants is an invariant itself, it can be shown that $CI$ is an invariant characterising all components.

**Proposition 2** *$CI \stackrel{\triangle}{=} \big(CI(C) \wedge \forall i.CI(P^i)\big)$ is an invariant of $C\|_\gamma^n P^i$.*

### 4.2 History Clocks & Auxiliary Constraints

A direct application of the rule (VR) on PTSs may be too weak in the sense that the component and the interaction invariants alone are usually not enough to prove global properties, especially when such properties involve relations between clocks in different components. Though component invariants encode timings of local clocks, because the interaction invariant is orthogonal on timing aspects, there is no direct way to constrain the bounds on the differences between clocks in different components. History clocks allow to decouple the analysis for components and for their composition. They make it possible to derive new global constraints from the simultaneity of interactions and the synchrony of time progress.

**Adding History Clocks.** History clocks are associated with actions and interactions. For a process $P$ we use $P^h$ to denote its extension with history clocks. The extension $C^h\|_\gamma^n {P^i}^h$ of $C\|_\gamma^n P^i$ is obtained from the extensions of the components alone together with the history clocks for interactions. As an illustration, Figure 4 shows the extension of the PTS in Figure 2a.

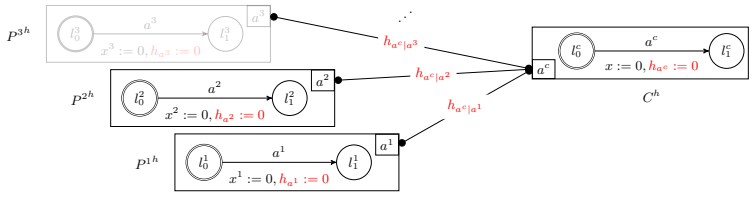



**Fig. 4.** Illustrating Components with History Clocks for (Inter)Actions

The mechanism of history clocks is as follows. When an interaction $\alpha$ takes place, the history clocks $h_\alpha$ and $h_a$ associated to $\alpha$ and to any action $a \in \alpha$ are reset. Thus they measure the time passed from the last occurrence of $\alpha$, respectively of $a$. Since there is no timing constraint involving history clocks, the behaviour of the components is not changed by the addition of history clocks, a fact which is shown by a similar argument as in [10].

**Proposition 3** *$C\|_\gamma^n P^i$ and $C^h\|_\gamma^n {P^i}^h$ are bisimilar.*

**Generating Interaction Equalities from History Clocks.** History clocks are introduced with the purpose of obtaining stronger invariants. Intuitively, the strengthening comes from the following observation. Each time an interaction $\alpha$ is executed, $h_\alpha$ and all the history clocks corresponding to the actions participating in $\alpha$ are reset synchronously, and then remain unchanged and *equal* until the next interaction is executed. Moreover, a history clock $h_a$ for an action $a$ from a last executed interaction $\alpha$ is necessarily *less* than any $h_\beta$ with $\beta$ another interaction containing $a$. This is because the clocks of the actions in $\alpha$ are the last ones being reset. Consequently, given a common action $a$ of $\alpha_1, \alpha_2, \ldots, \alpha_p$, $h_a$ is the minimum of $h_{\alpha_i}$, $h_a = \min_{i \in [p]} h_{\alpha_i}$.

In the parameterised case, the above observation is captured as follows. Each interaction pattern $\alpha$ and each $a \in \alpha$ are associated to the arrays $\overline{h_\alpha}$, respectively $\overline{h_a}$. Let $\alpha$ be of the form $(\ldots, (a, o), \ldots)$. For a given index $i$, $a^i$ appears in $\alpha^{i-o}$. Consequently, $\overline{h_a}[i]$ is the minimum among $\overline{h_\alpha}[i-o]$:

$$\mathcal{E}(a^i) = \left( \bigvee_{(a,o)\in\alpha} \overline{h_a}[i] = \overline{h_\alpha}[i-o] \right) \wedge \left( \bigwedge_{(a,o)\in\alpha} \overline{h_a}[i] \leq \overline{h_\alpha}[i-o] \right)$$

By switching perspective from that of $P^i$ to that of $C$, we obtain, for an action $a^c$ in $\Sigma^c$, the following quantified formula:

$$\mathcal{E}(a^c) = \exists j. \left( \bigvee_{a^c\in\alpha} h_{a^c} = \overline{h_\alpha}[j] \right) \wedge \forall i. \left( \bigwedge_{a^c\in\alpha} h_{a^c} \leq \overline{h_\alpha}[i] \right).$$

The existential quantifier is needed to express that $h_{a^c}$ is the minimum among an unbounded number of history clocks associated to interactions containing $a^c$.

To combine both perspectives, we define $\mathcal{E}(\gamma) = \forall i \wedge_a \mathcal{E}(a^i) \wedge_{a^c} \mathcal{E}(a^c)$. By an inductive argument, it can be shown that these constraints are an invariant.

**Proposition 4** *$\mathcal{E}(\gamma)$ is an invariant of $C^h \|_\gamma^n P^{ih}$.*

*Example 6.* For the star topology $\gamma = \{\alpha\}$ with $\alpha = \big(a^c, (a, 0)\big)$ we have that:

$$\begin{aligned}\mathcal{E}(\gamma) = \forall i.(\overline{h_a}[i] = \overline{h_\alpha}[i] \wedge \overline{h_a}[i] \leq \overline{h_\alpha}[i]) \wedge \\ \exists j.(h_{a^c} = \overline{h_\alpha}[j]) \wedge \forall i.(h_{a^c} \leq \overline{h_\alpha}[i])\end{aligned}$$

As for the ring topology $\gamma = \{\alpha\}$ with $\alpha = \big((s, 0), (r, 1)\big)$ we have:

$$\mathcal{E}(\gamma) = \forall i.\big(\overline{h_s}[i] = \overline{h_\alpha}[i] \wedge \overline{h_s}[i] \leq \overline{h_\alpha}[i]\big) \wedge \big(\overline{h_r}[i] = \overline{h_\alpha}[i-1] \wedge \overline{h_r}[i] \leq \overline{h_\alpha}[i-1]\big).$$

**Generating Inequalities from Conflicting Interactions.** The equality constraints shown previously allow to relate local constraints obtained separately from the component invariants. Without conflicts, that is, when interactions do not share any action, the generated invariants are quite tight in the sense that $\mathcal{E}(\gamma)$ is essentially a conjunction of equalities. However, $\mathcal{E}(\gamma)$ is weaker in

the presence of conflicts because any action in conflict can be used in different interactions. The disjunctions in $\mathcal{E}(\gamma)$ reflect precisely this uncertainty. History clocks on interactions are introduced to capture the time lapses between conflicting interactions. The basic information we exploit is that when two conflicting interactions compete for the same action $a$, no matter which one is first, the other one must wait until the component which owns $a$ is again able to execute $a$. This is referred to as a "separation constraint" for conflicting interactions. Since we make the distinction between the actions in $C$ and $P$, the reasoning goes as for $\mathcal{E}$, by a case distinction:

$$\mathcal{S}(a^c) = \forall i_1, i_2. \bigwedge_{\substack{\alpha \ni a^c \\ \beta \ni a^c \\ i_1 \neq i_2 \vee \alpha \neq \beta}} \left|\overline{h_\alpha}[i_1] - \overline{h_\beta}[i_2]\right| \geq k_{a^c}$$

$$\mathcal{S}(a^i) = \bigwedge_{\substack{(a,o_1) \in \alpha \\ (a,o_2) \in \beta \\ o_1 \neq o_2 \vee \alpha \neq \beta}} \left|\overline{h_\alpha}[i - o_1] - \overline{h_\beta}[i - o_2]\right| \geq k_a$$

where $k_{a^c}$, $k_a$ are lower bounds of the time elapsed between two consecutive executions of $a^c$ in $C$, respectively of $a$ in $P$, bounds which can be statically computed from the timed automata of $C$, respectively of $P$. Similarly to $\mathcal{E}(\gamma)$, $\mathcal{S}(\gamma)$ is defined by combining $\mathcal{S}(a^c)$ and $\mathcal{S}(a^i)$: $\mathcal{S}(\gamma) = \forall i \wedge_a \mathcal{S}(a^i) \wedge_{a^c} \mathcal{S}(a^c)$. Furthermore, $\mathcal{S}(\gamma)$ can be shown to be an invariant.

**Proposition 5** *$\mathcal{S}(\gamma)$ is an invariant of $C^h \|^n_\gamma P^{i\,h}$.*

*Example 7.* For the star topology $\gamma = \{\alpha\}$ with $\alpha = \big(a^c, (a, 0)\big)$ we have that:

$$\mathcal{S}(\gamma) = \forall i_1, i_2. \left|\overline{h_\alpha}[i_1] - \overline{h_\alpha}[i_2]\right| \geq k_{a^c} \tag{3}$$

As the ring topology $\gamma = \{\alpha\}$ with $\alpha = \big((s, 0), (r, 1)\big)$ does not have conflicts, for illustration purposes, we consider the following slight variation $\alpha = \big((r, 0), (s, 1), (r, 2)\big)$ corresponding to sends being forwarded to the left and to the right. In this case, we have that:

$$\mathcal{S}(\gamma) = \forall i. \left|\overline{h_\alpha}[i] - \overline{h_\alpha}[i - 2]\right| \geq k_r \tag{4}$$

with $k_r$ being the lower bound of the time elapsed between two consecutive $r$.

### 4.3 Interaction Invariants

Interaction invariants $II(\gamma)$ are induced by the synchronisations and have the form of global conditions involving control locations of components. Previous work considered boolean conditions [13] as well as linear constraints [28] as methods for generating $II(\gamma)$. These approaches do not easily generalise to the parameterised case: applying the method of [13] boils down to transforming to conjunctive normal forms quantified formulae while the one in [28] boils down to solving

an unbounded number of equations. Our solution is to adopt a $k$-window abstraction instead. To obtain such an abstraction, the main step is to generate interactions involving only actions from $\Sigma_i$ with $i \leq k$. Let $\alpha$ be an interaction pattern $\big(a^c, (a_1, o_1), \ldots, (a_m, o_m)\big)$. Recall that the offsets $o_i$ are in ascending order. We define $gen(\alpha, k)$ as $\cup_{i\in[k]} proj(\alpha^i)$ where $proj(\alpha^i) = \big(a^c, a_1^i, a_2^{i+o_2}, \ldots, a_j^{i+o_j}\big)$ and $j$ is the last index for which $i + o_j \leq k$. We recall that addition is taken modulo $n$. We denote $\cup_{\alpha\in\gamma} gen(\alpha, k)$ by $\gamma_k$. Given this construction, the remaining steps for computing a $k$-window abstraction are:

1. use the above mentioned methods or simply compute the set of the reachable states of $C$ interacting with $k$ processes $P^i$ to generate $II_k \stackrel{\triangle}{=} II(\gamma_k)$;
2. reindex $II_k$ by renaming all indices $j \in [k]$ to $j + i$ to obtain $II_k^*$ of the form $\forall i. II_k[j \leftarrow j + i]$.

We note that $k$-window is an abstraction of the original system $C\|_\gamma^n P^i$. Consequently, each invariant computed with respect to the $k$-window is also an invariant of $C\|_\gamma^n P^i$.

**Proposition 6** *The formula $(k < n \vee II_k^*)$ is an invariant of $C\|_\gamma^n P^i$.*

*Example 8.* We consider the star topology present in the toy example shown in Figure 2a. If we abstract to a window of size 1, the first step consists in the computation of the interaction invariant for $C$ interacting with $P^1$, where the interaction set after projection is $\gamma_1 = \{(a^c, a^1), a^c\}$. Using the reachable set of $C\|_{\gamma_1} P^1$, the interaction invariant for this abstraction is $II_1 = \overline{l_0}[1] = 1 \vee l_1^c = 1$. By Step 2, the interaction invariant for $C\|_\gamma^n P^i$ is $II^* = \forall i. \overline{l_0}[i] = 1 \vee l_1^c = 1$.

### 4.4 Parameterised (VR)

Taking into account the clock constraints $\mathcal{E}$ and $\mathcal{S}$, the generalisation of the rule (VR) recalled in the introduction to the parameterised case boils down to checking the validity of the following formula:

$$\underbrace{CI \wedge (k < n \vee II_k^*) \wedge \mathcal{E}(\gamma) \wedge \mathcal{S}(\gamma)}_{GI} \rightarrow \Psi \tag{5}$$

or equally the unsatisfiability of $GI \wedge \neg\Psi$. These formulae are T-assertions whenever $\Psi$ is a T-assertion itself.

**Proposition 7** *For $\Psi$ a T-assertion, $GI \rightarrow \Psi$ is a T-assertion itself.*

*Proof (sketch).* In prenex normal form, each invariant is a T-assertion. We only detail the more interesting cases of $\mathcal{E}$ and $\mathcal{S}$:

$$\begin{aligned}\mathcal{E}(\gamma) &= \forall i \wedge_a \mathcal{E}(a^i) \wedge_{a^c} \mathcal{E}(a^c)\\ &\equiv \forall i_1, i_2. \exists j_{\Sigma^c}. \left(\bigvee_{(a,o)\in\alpha} \overline{h_a}[i_1] = \overline{h_\alpha}[i_1 - o]\right) \wedge \left(\bigwedge_{(a,o)\in\alpha} \overline{h_a}[i_1] \leq \overline{h_\alpha}[i_1 - o]\right) \wedge\\ &\qquad \wedge_{a^c} \left(\bigvee_{a^c\in\alpha} h_{a^c} = \overline{h_\alpha}[j_{a^c}]\right) \wedge \left(\bigwedge_{a^c\in\alpha} h_{a^c} \leq \overline{h_\alpha}[i_2]\right)\end{aligned} \tag{6}$$

$$\begin{aligned}\mathcal{S}(\gamma) &= \forall i.\, \wedge_a \mathcal{S}(a^i) \wedge_{a^c} \mathcal{S}(a^c)\\ &\equiv \forall i_1, i_2.\, \wedge_a \bigwedge_{\substack{(a,o_1)\in\alpha\\(a,o_2)\in\beta\\o_1\neq o_2 \vee \alpha\neq\beta}} \left|\overline{h_\alpha}[i-o_1] - \overline{h_\beta}[i-o_2]\right| \geq k_a\\ &\qquad \wedge_{a^c} \bigwedge_{\substack{\alpha\ni a^c\\ \beta\ni a^c}} \left|\overline{h_\alpha}[i_1] - \overline{h_\beta}[i_2]\right| \geq k_{a^c}\end{aligned}$$

where $a, a^c$ are arbitrary actions in $\Sigma$, respectively $\Sigma^c$, $\exists j_{\Sigma^c}$ denotes $\exists j_{a_1} j_{a_2} \ldots j_{a_m}$ for $\Sigma^c = \{a_1, \ldots, a_m\}$ and $j_{a^c}$ stands for an arbitrary element in $j_{\Sigma^c}$.

Observing that all quantified variables are not shared among invariants, we can rename these such that there are no overlappings and use the following basic equivalences where *op* denotes any logical connective and $\mathbf{Q}$ any quantifier:

$$\begin{aligned}&\mathbf{Q}x\mathbf{Q}y.(P(x)\ op\ R(y)) \equiv \mathbf{Q}y\mathbf{Q}x.(P(x)\ op\ R(y))\\ &P\ op\ \mathbf{Q}y.R(y) \equiv \mathbf{Q}y.(P\ op\ R(y))\end{aligned}$$

to finally transform $GI \to \Psi$ itself into a T-assertion. □

Proposition 7 allows us to apply the small model theorem from Section 2.

**Corollary 1.** *For a PTS* $C\|^n_\gamma P^i$ *and a global property* $\Psi \stackrel{\triangle}{=} \forall \bar{s} \exists \bar{t}.\Psi^\circ$ *it is enough to check the validity of* $\neg GI \vee \Psi$ *for* $n \leq \#\bar{s} + \#\Sigma^c + 2$ *in order to assert the validity of* $\Psi$ *for any* $n$.

*Proof.* By Proposition 1, the bound depends on the number of universally quantified variables in $\Psi$ and on the size of $\Sigma^c$, by Equation (6). The latter is the number of universal quantifiers in $\neg GI$.

*Example 9.* As an illustration, we work through the toy example from head to tail. As a safety state property we take $\Psi \stackrel{\triangle}{=} \exists i.x^c = \overline{x}[i]$, that is, $\Psi$ expresses that one of the clocks in $\bar{x}$ has the same value as $x^c$. We have already gone through the main ingredients with Equations (1)-(3) in Examples 5-8. What is left is to combine them and rename the quantified variables to obtain:

$$\begin{aligned}\forall i\ \exists j_1, j_2, j_3, j_4, j_5.\Bigg(&\neg\Big((l_0^c \wedge h_{a^c} \geq 0 \wedge x^c \geq 0 \vee l_1^c \wedge x^c = h_{a^c} \geq 0) \wedge\\ &(\overline{l_0}[j_1] \wedge \overline{h_a}[j_1] \geq 0 \wedge \overline{x}[j_1] \geq 0 \vee \overline{l_1}[j_1] \wedge \overline{x}[j_1] = \overline{h_a}[j_1] \geq 0) \wedge\\ &(\overline{h_a}[j_2] = \overline{h_\alpha}[j_2] \geq h_{a^c}) \wedge (h_{a^c} = \overline{h_\alpha}[i]) \wedge \left|\overline{h_\alpha}[j_3] - \overline{h_\alpha}[j_5]\right| \geq k_{a^c}\Big) \vee\\ &x^c = \overline{x}[j_4]\Bigg)\end{aligned} \tag{7}$$

Above, we have used the component invariants with respect to the extensions with history clocks[6]. By applying Corollary 1, we can assert the correctness of $\Psi$ from the validity the formula in (7) for $n \leq 3$ processes.

[6] The computation is the same as the one in Example 5. For illustration, we only show the component invariant for $C^h$: $CI(C^h) = l_0^c \wedge h_{a^c} \geq 0 \wedge x^c \geq 0 \vee l_1^c \wedge x^c = h_{a^c} \geq 0$.

## 5 Experiments

To illustrate the star and the ring topologies, we take the following case studies.

**Train gate controller:** This is the parameterised version of the classical example from [9]. The system is depicted in Figure 5. It is composed of a controller, a gate and an arbitrary number of trains. The controller lowers (raises) the gate when a train enters (exits). The property $\Psi$ is that the gate enters $g_1$ location only if one of the trains left $far$ location: $\exists i.\neg\overline{far}[i] \vee \neg g_1$. For this example,

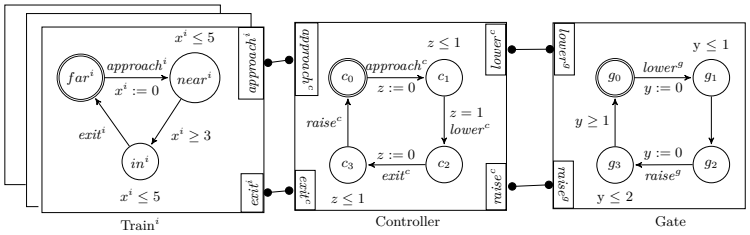



**Fig. 5.** A Controller Interacting with an Arbitrary Number of Trains and a Gate

$II(\gamma)$ plays no role: $\mathcal{E}(\gamma)$ and $\mathcal{S}(\gamma)$ are enough to check the validity of $\neg GI \vee \Psi$ for the bound of 3. The small model result justifies this check as sufficient. As a side note, additionally, we proved deadlock-freedom (the bound was 5).

**Token Ring:** The protocol depicted in Figure 6 is an adaption from [33]. Every process $P^i$ receives the token from $P^{i-1}$ through the interaction $(s^{i-1}, r^i)$. It then moves to $t_1^i$ location and after passing the token, it moves from $t_2^i$ to $a^i$. Once $P^i$ sends the token, it cannot have it again before 2 time units. This constraint is expressed using clock $x^i$. Initially, $P^1$ is in $t_1^1$, meaning that it possesses the token, while all other $P^i$ are at location $a^i$, waiting for the reception of the token. The property $\Psi$ is that one and only one process possesses the token: $\Psi \stackrel{\triangle}{=} \exists i.\forall j \neq i.\big(\neg\overline{a}[i] \wedge \overline{a}[j]\big)$.

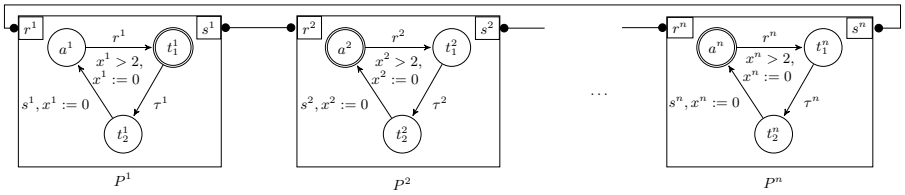



**Fig. 6.** An Arbitrary Number of Processes in a Ring Topology

For this example, the number of universal quantifiers in $\neg GI \vee \Psi$ is 3. As a more interesting observation from our experiments, we add that the interaction invariant as computed automatically in [10] has the form of a T-assertion: $\exists j.\,(\overline{t_1}[j] \vee \overline{t_2}[j])$. What it expresses is that at least one $P^i$ is not at $\overline{a}[i]$ location, or equally, that the token is not lost. This invariant is, along with clock constraints from $CI(P^i)$, and $\mathcal{E}(\gamma)$ as $\forall i.\overline{h_s}[i] = \overline{h_\alpha}[i] \wedge \overline{h_r}[i] = \overline{h_\alpha}[i-1]$, necessary to show that exactly one $P^i$ is at $\overline{a}[i]$ at a given time.

## 6 Conclusion & Future Work

We have presented a compositional method for the verification of parameterised timed systems. The key element we made use of is a typical small model theorem. The small model theorem does not hold in the context of networks of *parametric* timed automata in its most general case. If the particular case of timed automata with parameter $n$ can be handled by extending the fragment of T-assertions with results from [24], this is no longer the case for timed automata parametric in their indices. This is because the invariant of such timed automata would involve constraints of type $x_i \geq i \cdot ct$ and such constraints are not allowed by the grammar of T-assertions. It would be of interest to investigate in this direction if possible extensions of $T$-assertions are foreseeable. Another possible alternative is to exploit the inherent symmetry in such systems.

Besides showing how compositional verification can benefit from small model theorems, we have also shown the close relation between interactions and topologies. In this respect, we note that tree-like topologies are more tricky to encode: offsets as constants are too weak but we intuit that the offset would need to contain offsets itself. To allow more sophisticated interaction patterns, we could also borrow some of the constructions in [24] to express constraints like periodicity on indices. That is, given an interaction pattern $\alpha$, instead of generating $\alpha^i$ for $i \in [n]$, it would be of interest to generate $\alpha^i$ only for indices $i$ satisfying a constraint like parity, or boundedness.

A third possible extension we will consider is with respect to false positives: as any incomplete method, (VR) may yield spurious counterexamples. We will look into how counterexample-based refinement techniques can in turn be applied in the context of (parameterised) timed systems. Given that the search space (of reals) is infinite, the main difficulty we envisage is the generalisation of the concrete real values from a given counterexample to a more generic characterisation which would guarantee convergence.

## References


1. P. A. Abdulla, K. Cerans, B. Jonsson, and Y. Tsay. General decidability theorems for infinite-state systems. In *LICS*, 1996.
2. P. A. Abdulla, G. Delzanno, O. Rezine, A. Sangnier, and R. Traverso. On the verification of timed ad hoc networks. In *FORMATS*, 2011.
3. P. A. Abdulla, J. Deneux, and P. Mahata. Closed, open, and robust timed networks. *ENTCS*, 138(3), 2005.
4. P. A. Abdulla and B. Jonsson. On the existence of network invariants for verifying parameterized systems. In *Correct System Design*, 1999.
5. P. A. Abdulla and B. Jonsson. Model checking of systems with many identical timed processes. *Theor. Comput. Sci.*, 290(1), 2003.
6. P. A. Abdulla, B. Jonsson, M. Nilsson, and M. Saksena. A survey of regular model checking. In *CONCUR*, 2004.
7. R. Alur. Timed automata. In *CAV*, 1999.
8. R. Alur, C. Courcoubetis, D. L. Dill, N. Halbwachs, and H. Wong-Toi. An implementation of three algorithms for timing verification based on automata emptiness. In *RTSS*, 1992.

9. R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 1994.
10. L. Astefanoaei, S. B. Rayana, S. Bensalem, M. Bozga, and J. Combaz. Compositional invariant generation for timed systems. In *TACAS*, 2014.
11. K. Baukus, S. Bensalem, Y. Lakhnech, and K. Stahl. Abstracting ws1s systems to verify parameterized networks. In *TACAS*, 2000.
12. K. Baukus, K. Stahl, S. Bensalem, and Y. Lakhnech. Networks of processes with parameterized state space. *ENTCS*, 50(4), 2001.
13. S. Bensalem, M. Bozga, J. Sifakis, and T.-H. Nguyen. Compositional verification for component-based systems and application. In *ATVA*, 2008.
14. A. Bouajjani, Y. Jurski, and M. Sighireanu. A generic framework for reasoning about dynamic networks of infinite-state processes. In *TACAS*, 2007.
15. R. Bruttomesso, A. Carioni, S. Ghilardi, and S. Ranise. Automated analysis of parametric timing-based mutual exclusion algorithms. In *NFM*, 2012.
16. A. Carioni, S. Ghilardi, and S. Ranise. Mcmt in the land of parametrized timed automata. In *VERIFY@IJCAR*, 2010.
17. E. A. Emerson and V. Kahlon. Reducing model checking of the many to the few. In *CADE*, 2000.
18. E. A. Emerson and K. S. Namjoshi. Reasoning about rings. In *POPL*, 1995.
19. E. A. Emerson and A. P. Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1/2), 1996.
20. A. Finkel. A generalization of the procedure of karp and miller to well structured transition systems. In *ICALP*, 1987.
21. A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2), 2001.
22. S. M. German and A. P. Sistla. Reasoning about systems with many processes. *J. ACM*, 39(3), 1992.
23. S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Towards smt model checking of array-based systems. In *IJCAR*, 2008.
24. P. Habermehl, R. Iosif, and T. Vojnar. What else is decidable about integer arrays? In *FOSSACS*, 2008.
25. T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Inf. Comput.*, 1994.
26. C. Ihlemann, S. Jacobs, and V. Sofronie-Stokkermans. On local reasoning in verification. In *TACAS*, 2008.
27. T. T. Johnson and S. Mitra. A small model theorem for rectangular hybrid automata networks. In *FMOODS/FORTE*, 2012.
28. A. Legay, S. Bensalem, B. Boyer, and M. Bozga. Incremental generation of linear invariants for component-based systems. In *ACSD*, 2013.
29. D. Lesens, N. Halbwachs, and P. Raymond. Automatic verification of parameterized linear networks of processes. In *POPL*, 1997.
30. D. Lesens, N. Halbwachs, and P. Raymond. Automatic verification of parameterized networks of processes. *Theor. Comput. Sci.*, 256(1-2), 2001.
31. K. S. Namjoshi. Symmetry and completeness in the analysis of parameterized systems. In *VMCAI*, 2007.
32. A. Pnueli, S. Ruah, and L. D. Zuck. Automatic deductive verification with invisible invariants. In *TACAS*, 2001.
33. J. Reich. Processes, roles and their interactions. In *Proceedings of IWIGP*, 2012.
34. P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants. In *AVMFSS*, 1989.
35. W. Yi, P. Pettersson, and M. Daniels. Automatic verification of real-time communicating systems by constraint-solving. In *FORTE*, 1994.